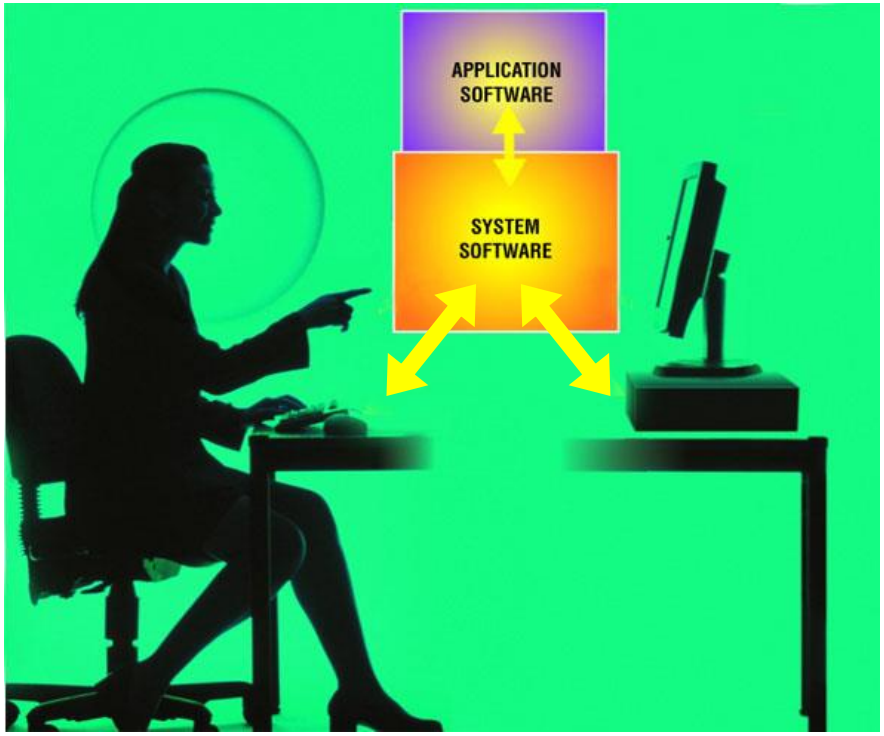
Chapter 1 Background

Outlines

- Brief introduction to system software
- System software and machine architecture
- Simplified Instructional Computer (SIC)
  - Architecture
  - Assembly language
- Architecture of real machines
  - CISC
  - RISC

Introduction

- Purpose of system software: users or application software can focus on problem solving, without needing to know how the machine works internally.
- System software:
  - Operating system
  - Text editor
  - Compiler
  - Assembler
  - Linker
  - Loader
  - debugger



Introduction

- This course aims at:
  - Understanding what is going on “behind the scenes”
  - The design and implementation of system software, such as
    - Assemblers
    - Loaders and linkers
    - Macro processors
    - Compilers
    - Operating systems

System Software and Architecture

- Machine dependency of system software
  - System programs are intended to support the operation and use of the computer.
  - Machine architecture differs in:
    - Machine code
    - Instruction formats
    - Addressing mode
    - Registers
- Machine independency of system software
  - General design and logic is basically the same:
    - Code optimization
    - Subprogram linking

System Software and Architecture

- System software will be discussed:
  1. The basic functions
  2. Machine-dependent functions
  3. Machine-independent functions
  4. Design options (single-pass vs. multi-pass)
  5. Examples of implementations

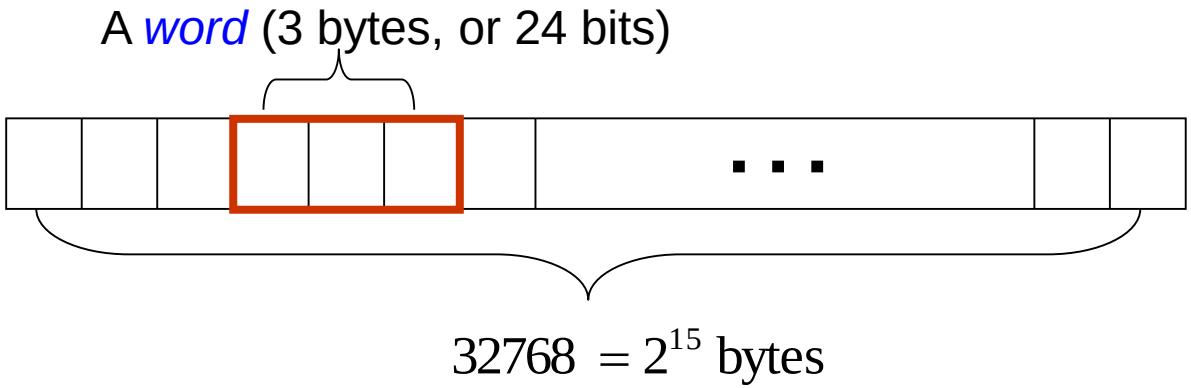
Simplified Instructional Computer

SIC and SIC/XE

- Why the simplified instructional computer
  - To avoid various unique features and idiosyncrasies of a particular machine
  - To focus on central, fundamental, and commonly encountered features and concepts
- Two versions
  - Standard model (SIC) and an XE version (SIC/XE)
  - Upward compatible
    - Programs for SIC can run on SIC/XE

SIC Architecture

- Memory



SIC Architecture

- Registers
  - Five 24-bit registers

Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
3-7: reserved for future use		
PC	8	Program counter; contains the address of the next instruction to be fetched for execution
SW	9	Status word; contains a variety of information, including a Condition Code (CC)

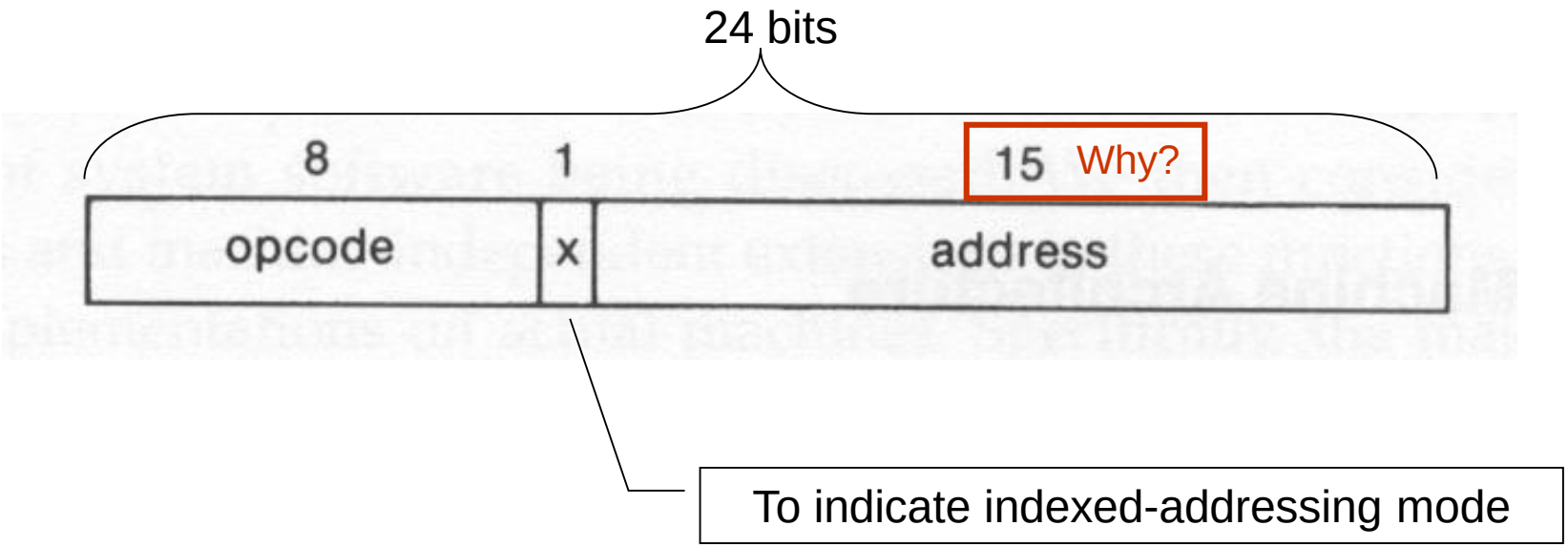
SIC Architecture

- Data formats
  - Characters: 8-bit ASCII codes
  - Integers:
    - 24-bit binary numbers
    - 2's complement for negative values
  - Floating-point numbers: No

Decimal	Signed-magnitude	Signed-1's complement	Signed-2's complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	X
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001
-8	X	X	1000

SIC Architecture

- Instruction formats



SIC Architecture

- Addressing modes

Mode	Indication	Target address calculation
Direct	x = 0	TA = address
Indexed	x = 1	TA = address + (X)

( ): Contents of a register or a memory location

SIC Architecture

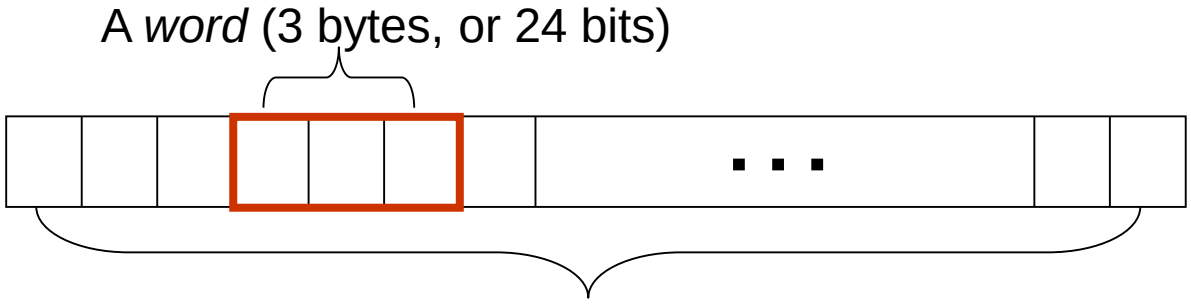
- Instruction set
  - Load and store registers
    - LDA, LDX, STA, STX
  - Integer arithmetic operations (involve register A and a word in memory, save result in A)
    - ADD, SUB, MUL, DIV
  - Comparison (involves register A and a word in memory, save result in the condition code (CC) of SW)
    - COMP
  - Conditional jump instructions (according to CC)
    - JLT, JEQ, JGT
  - Subroutine linkage
    - JSUB (jumps and places the return address in register L)
    - RSUB (returns to the address in L)

SIC Architecture

- Input and output
  - Each IO device is assigned a unique 8-bit code
  - One byte at a time to or from the rightmost 8 bits of register A
  - Three instructions:
    - Test device (TD):
      - Test whether the device is ready to send/receive
      - Test result is set in CC
    - Read data (RD): read one byte from the device to register A
    - Write data (WD): write one byte from register A to the device

SIC/XE Architecture

- Memory: from 32768 to 1 mega



$$1\,\text{megabyte} = 2^{20}\ \text{bytes}$$

SIC/XE Architecture

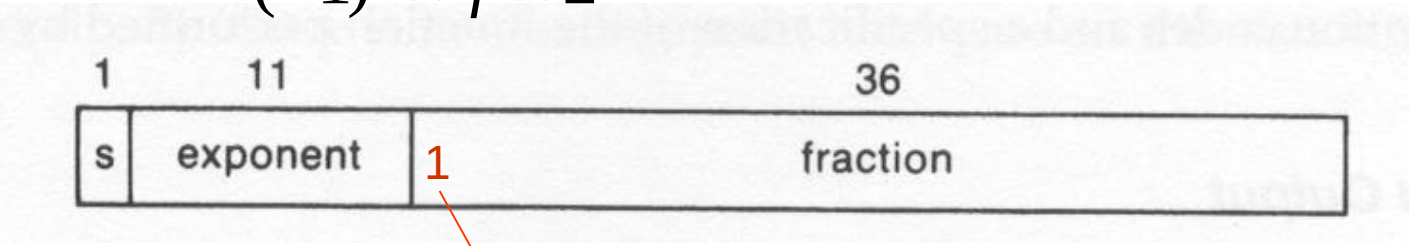
- Registers: 4 additional registers

Mnemonic	Number	Special use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
PC	8	Program counter; contains the address of the next instruction to be fetched for execution
SW	9	Status word; contains a variety of

Mnemonic	Number	Special use
B	3	Base register; used for addressing
S	4	General working register—no special use
T	5	General working register—no special use
F	6	Floating-point accumulator (48 bits)

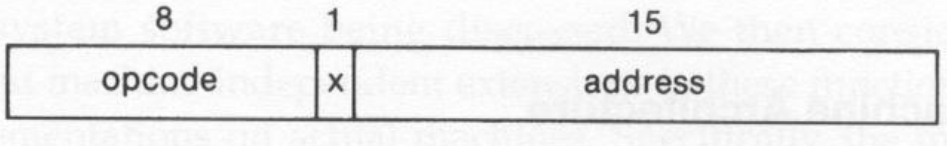
SIC/XE Architecture

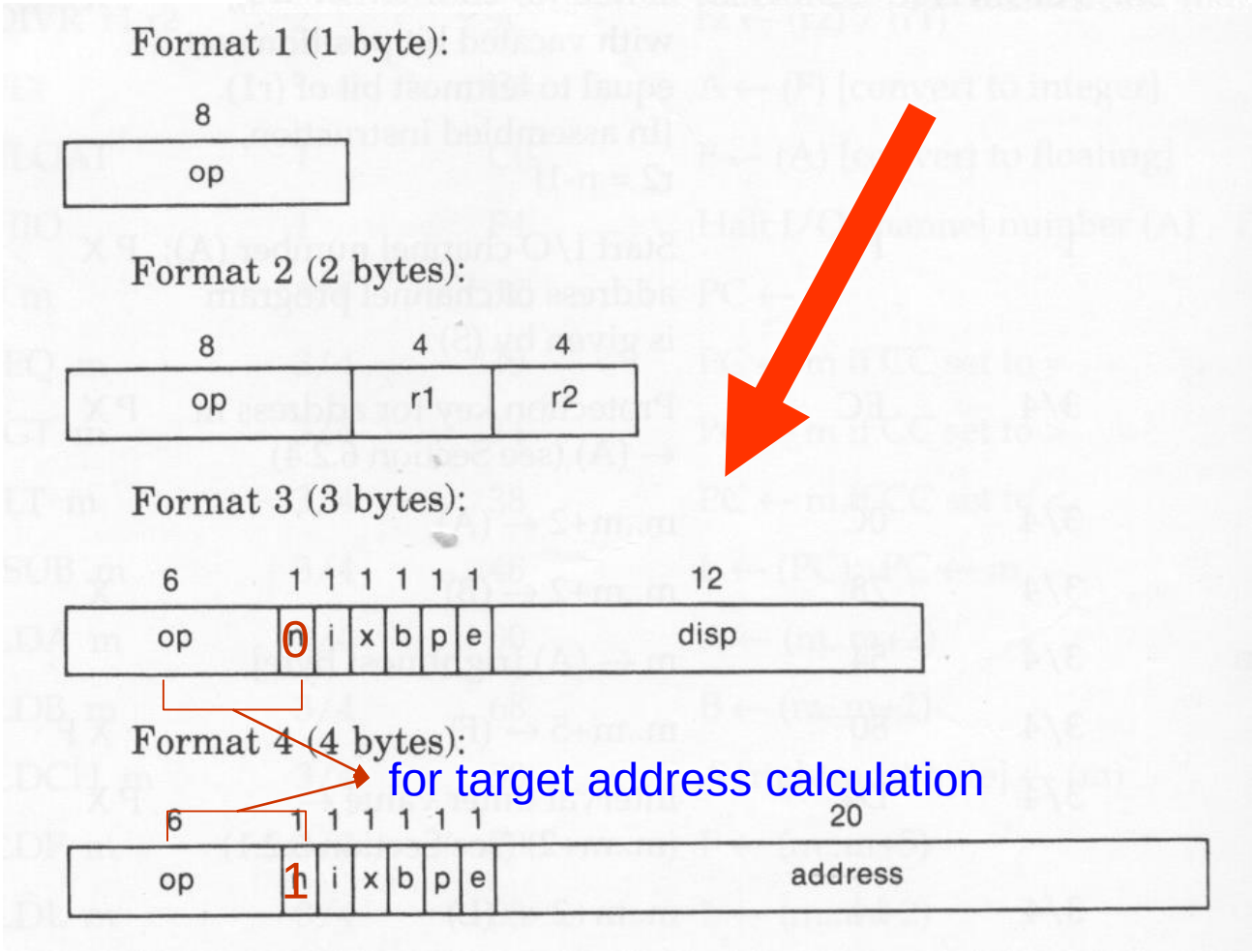
- Data formats
  - Characters: 8-bit ASCII codes
  - Integers:
    - 24-bit binary numbers
    - 2's complement for negative values
  - Floating-point numbers (48 bits)
    - sign bit *s*
    - fraction *f*: a value between 0 and 1
    - exponent *e*: unsigned binary number between 0 and 2047
    - value: $(-1)^s \times f \times 2^{e-1024}$



SIC/XE Architecture

- Instruction formats





No memory reference

Relative addressing

Extended address field

SIC/XE Architecture

- Addressing modes:
  - two new relative addressing for format 3

Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	(0 ≤ disp ≤ 4095)
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	(–2048 ≤ disp ≤ 2047)

  - direct addressing for formats 3 and 4 if b=p=0
  - Indexed addressing can be combined if x=1:
    - the term (x) should be added

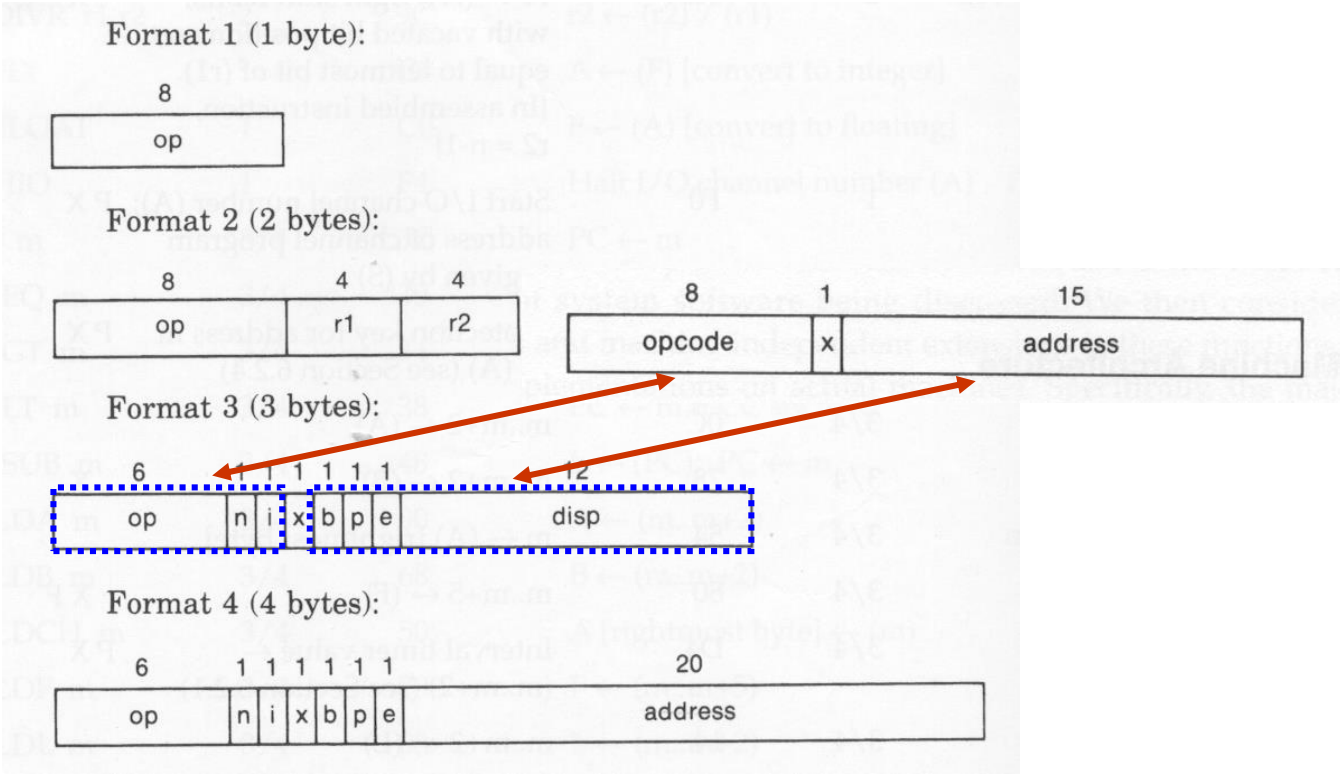
Mode	Indication	Target address calculation
Direct	x = 0	TA = address
Indexed	x = 1	TA = address + (X)

SIC/XE Architecture

- Bits x,b,p,e: how to calculate the target address
  - relative, direct, and indexed addressing modes
- Bits i and n: how to use the target address (TA)
  - i=1, n=0: immediate addressing
    - TA is used as the operand value, no memory reference
  - i=0, n=1: indirect addressing
    - The word at the TA is fetched
    - Value in this word is taken as the address of the operand value
  - i=0, n=0 (in SIC), or
  - i=1, n=1 (in SIC/XE): simple addressing
    - TA is taken as the address of the operand value
- Exception: indexing cannot be used with immediate or indirect addressing modes.

SIC/XE Architecture

- For upward compatibility
  - 8-bit binary codes for all SIC instructions end in 00
  - If n=i=0, bits b,p,e are considered as part of the 15-bit address field



Addressing type	Flag bits n i x b p e	Assembler language notation	Calculation of target address TA	Operand	Notes
Simple	1 1 0 0 0 0	op c	disp	(TA)	D
	1 1 0 0 0 1	+op m	addr	(TA)	4 D
	1 1 0 0 1 0	op m	(PC) + disp	(TA)	A
	1 1 0 1 0 0	op m	(B) + disp	(TA)	A
	1 1 1 0 0 0	op c,X	disp + (X)	(TA)	D
	1 1 1 0 0 1	+op m,X	addr + (X)	(TA)	4 D
	1 1 1 0 1 0	op m,X	(PC) + disp + (X)	(TA)	A
	1 1 1 1 0 0	op m,X	(B) + disp + (X)	(TA)	A
	0 0 0 - - -	op m	b/p/e/disp	(TA)	D S
	0 0 1 - - -	op m,X	b/p/e/disp + (X)	(TA)	D S
Indirect	1 0 0 0 0 0	op @c	disp	((TA))	D
	1 0 0 0 0 1	+op @m	addr	((TA))	4 D
	1 0 0 0 1 0	op @m	(PC) + disp	((TA))	A
	1 0 0 1 0 0	op @m	(B) + disp	((TA))	A
Immediate	0 1 0 0 0 0	op #c	disp	TA	D
	0 1 0 0 0 1	+op #m	addr	TA	4 D
	0 1 0 0 1 0	op #m	(PC) + disp	TA	A
	0 1 0 1 0 0	op #m	(B) + disp	TA	A

c: constant between 0 and 4095
m: memory address or constant larger than 4095

4: Format 4

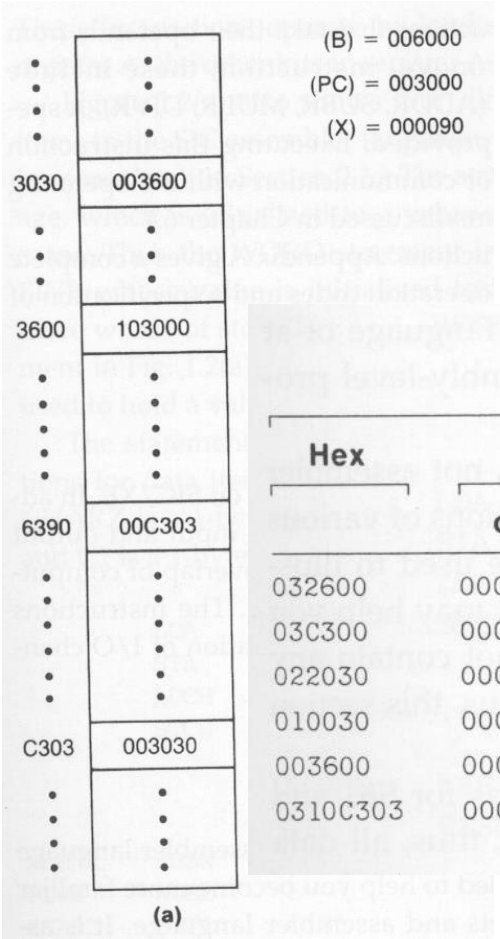
A: Relative addressing

D: Direct addressing

S:Compatible with SIC

SIC/XE Architecture

- Examples of addressing mode for LDA



(a)

Machine instruction									Target address	Value loaded into register A
Hex	Binary									
	op	n	i	x	b	p	e	disp/address		
032600	000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0	0000 0011 0000	3030	103000
010030	000000	0	1	0	0	0	0	0000 0011 0000	30	000030
003600	000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

(b)

SIC/XE Architecture

- Instruction set
  - Load and store registers
    - LDA, LDX, STA, STX, LDB, STB, …
  - Integer arithmetic operations
    - ADD, SUB, MUL, DIV, ADDF, SUBF, MULF, DIVF, ADDR, SUBR, MULR, DIVR
  - Comparison COMP
  - Conditional jump instructions (according to CC)
    - JLT, JEQ, JGT
  - Subroutine linkage
    - JSUB
    - RSUB
  - Register move
    - RMO
  - Supervisor call (for generating an interrupt)
    - SVC

SIC/XE Architecture

- Input and output
  - IO device
    - Three instructions:
      - Test device (TD)
      - Read data (RD)
      - Write data (WD)
  - IO channels
    - Perform IO while CPU is executing other instructions
    - Three instructions:
      - SIO: start the operation of IO channel
      - TIO: test the operation of IO channel
      - HIO: halt the operation of IO channel

I/O Mechanisms

- Polling I/O
- Interrupt-Driven I/O
- DMA (Direct Memory Access) I/O

SIC/XE Instruction Set

Mnemonic	Format	Opcode	Effect	Notes
ADD m	3/4	18	A ← (A) + (m..m+2)	
ADDF m	3/4	58	F ← (F) + (m..m+5)	X F
ADDR r1,r2	2	90	r2 ← (r2) + (r1)	X
AND m	3/4	40	A ← (A) & (m..m+2)	
CLEAR r1	2	B4	r1 ← 0	X
COMP m	3/4	28	(A) : (m..m+2)	C
COMPF m	3/4	88	(F) : (m..m+5)	X F C
COMPR r1,r2	2	A0	(r1) : (r2)	X C
DIV m	3/4	24	A ← (A) / (m..m+2)	
DIVF m	3/4	64	F ← (F) / (m..m+5)	X F
DIVR r1,r2	2	9C	r2 ← (r2) / (r1)	X
FIX	1	C4	A ← (F) [convert to integer]	X F
FLOAT	1	C0	F ← (A) [convert to floating]	X F
HIO	1	F4	Halt I/O channel number (A)	P X

X: only for XE

C: set CC

F: floating-point

P: privileged

J m	3/4	3C	PC ← m	
JEQ m	3/4	30	PC ← m if CC set to =	
JGT m	3/4	34	PC ← m if CC set to >	
JLT m	3/4	38	PC ← m if CC set to <	
JSUB m	3/4	48	L ← (PC); PC ← m	
LDA m	3/4	00	A ← (m..m+2)	
LDB m	3/4	68	B ← (m..m+2)	X
LDCH m	3/4	50	A [rightmost byte] ← (m)	
LDF m	3/4	70	F ← (m..m+5)	X F
LDL m	3/4	08	L ← (m..m+2)	
LDS m	3/4	6C	S ← (m..m+2)	X
LDT m	3/4	74	T ← (m..m+2)	X
LDX m	3/4	04	X ← (m..m+2)	
LPS m	3/4	D0	Load processor status from information beginning at address m (see Section 6.2.1)	P X
MUL m	3/4	20	A ← (A) * (m..m+2)	

for interrupt

Mnemonic	Format	Opcode	Effect	Notes
MULF m	3/4	60	F ← (F) * (m..m+5)	X F
MULR r1, r2	2	98	r2 ← (r2) * (r1)	X
NORM	1	C8	F ← (F) [normalized]	X F
OR m	3/4	44	A ← (A) | (m..m+2)	
RD m	3/4	D8	A [rightmost byte] ← data from device specified by (m)	P
RMO r1,r2	2	AC	r2 ← (r1)	X
RSUB	3/4	4C	PC ← (L)	
SHIFTL r1,n	2	A4	r1 ← (r1); left circular shift n bits. {In assembled instruction, r2 = n–1}	X
SHIFTR r1,n	2	A8	r1 ← (r1); right shift n bits, with vacated bit positions set equal to leftmost bit of (r1). {In assembled instruction, r2 = n-1}	X
SIO	1	F0	Start I/O channel number (A); address of channel program is given by (S)	P X

Set Storage Key for memory protection

SSK m	3/4	EC	Protection key for address m ← (A) (see Section 6.2.4)	P X
STA m	3/4	0C	m..m+2 ← (A)	
STB m	3/4	78	m..m+2 ← (B)	X
STCH m	3/4	54	m ← (A) [rightmost byte]	
STF m	3/4	80	m..m+5 ← (F)	X F
STI m	3/4	D4	Interval timer value ← (m..m+2) (see Section 6.2.1)	P X
STL m	3/4	14	m..m+2 ← (L)	
STS m	3/4	7C	m..m+2 ← (S)	X
STSW m	3/4	E8	m..m+2 ← (SW)	P
STT m	3/4	84	m..m+2 ← (T)	X
STX m	3/4	10	m..m+2 ← (X)	
SUB m	3/4	1C	A ← (A) – (m..m+2)	
SUBF m	3/4	5C	F ← (F) – (m..m+5)	X F

Mnemonic	Format	Opcode	Effect	Notes
SUBR r1,r2	2	94	r2 ← (r2) – (r1)	X
SVC n	2	B0	Generate SVC interrupt. {In assembled instruction, r1 = n}	X
TD m	3/4	E0	Test device specified by (m)	P C
TIO	1	F8	Test I/O channel number (A)	P X C
TIX m	3/4	2C	X ← (X) + 1; (X): (m..m+2)	C
TIXR r1	2	B8	X ← (X) + 1; (X): (r1)	X C
WD m	3/4	DC	Device specified by (m) ← (A) [rightmost byte]	P

SIC Programming Examples
(1) Data Movement

SIC

```
        LDA     FIVE        LOAD CONSTANT 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDCH    CHARZ       LOAD CHARACTER 'Z' INTO REGISTER A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
        .
        .
        .
ALPHA   RESW    1           ONE-WORD VARIABLE
FIVE    WORD    5           ONE-WORD CONSTANT
CHARZ   BYTE    C'Z'        ONE-BYTE CONSTANT
C1      RESB    1           ONE-BYTE VARIABLE
```

(a)

SIC/XE

```
        LDA     #5          LOAD VALUE 5 INTO REGISTER A
        STA     ALPHA       STORE IN ALPHA
        LDA     #90         LOAD ASCII CODE FOR 'Z' INTO REG A
        STCH    C1          STORE IN CHARACTER VARIABLE C1
        .
        .
        .
ALPHA   RESW    1           ONE-WORD VARIABLE
C1      RESB    1           ONE-BYTE VARIABLE
```

(b)

Assembler directives for defining storage

Address labels

Immediate addressing makes program faster due to fewer memory reference

SIC Programming Examples
(2) Arithmetic

SIC

```
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADD     INCR        ADD THE VALUE OF INCR
        SUB     ONE         SUBTRACT 1
        STA     DELTA       STORE IN DELTA
          .
          .
          .
ONE     WORD    1           ONE-WORD CONSTANT
.                           ONE-WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

BETA ← (ALPHA + INCR - 1)

DELTA ← (GAMMA + INCR - 1)

SIC Programming Examples
(2) Arithmetic

SIC/XE

```
        LDS     INCR        LOAD VALUE OF INCR INTO REGISTER S
        LDA     ALPHA       LOAD ALPHA INTO REGISTER A
        ADDR    S,A         ADD THE VALUE OF INCR
        SUB     #1          SUBTRACT 1
        STA     BETA        STORE IN BETA
        LDA     GAMMA       LOAD GAMMA INTO REGISTER A
        ADDR    S,A         ADD THE VALUE OF INCR
        SUB     #1          SUBTRACT 1
        STA     DELTA       STORE IN DELTA
        .
        .
        .
.                           ONE WORD VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

BETA ← (ALPHA + INCR - 1)
DELTA ← (GAMMA + INCR - 1)

SIC Programming Examples
(3) Looping and Indexing: part I

SIC

```
        LDX     ZERO        INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH    STR1,X      LOAD CHARACTER FROM STR1 INTO REG A
        STCH    STR2,X      STORE CHARACTER INTO STR2
        TIX     ELEVEN      ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT     MOVECH      LOOP IF INDEX IS LESS THAN 11
        .
        .
        .
STR1    BYTE    C'TEST STRING'   11-BYTE STRING CONSTANT
STR2    RESB    11               11-BYTE VARIABLE
.                                ONE-WORD CONSTANTS
ZERO    WORD    0
ELEVEN  WORD    11
```

Copy one 11-byte string to another

SIC Programming Examples
(3) Looping and Indexing: part I

SIC/XE

```
        LDT    #11          INITIALIZE REGISTER T TO 11
        LDX    #0           INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH   STR1,X       LOAD CHARACTER FROM STR1 INTO REG A
        STCH   STR2,X       STORE CHARACTER INTO STR2
        TIXR   T            ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT    MOVECH       LOOP IF INDEX IS LESS THAN 11
        .
        .
        .
STR1    BYTE   C'TEST STRING'   11-BYTE STRING CONSTANT
STR2    RESB   11               11-BYTE VARIABLE
```

Copy one 11-byte string to another

SIC Programming Examples
(4) Looping and Indexing: part II

SIC

```
        LDA     ZERO        INITIALIZE INDEX VALUE TO 0
        STA     INDEX
ADDLP   LDX     INDEX       LOAD INDEX VALUE INTO REGISTER X
        LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
        ADD     BETA,X      ADD WORD FROM BETA
        STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
        LDA     INDEX       ADD 3 TO INDEX VALUE
        ADD     THREE
        STA     INDEX
        COMP    K300        COMPARE NEW INDEX VALUE TO 300
        JLT     ADDLP       LOOP IF INDEX IS LESS THAN 300
        .
        .
        .
INDEX   RESW    1           ONE-WORD VARIABLE FOR INDEX VALUE
.                           ARRAY VARIABLES--100 WORDS EACH
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
.                           ONE-WORD CONSTANTS
ZERO    WORD    0
K300    WORD    300
THREE   WORD    3
```

GAMMA [ ] ← ALPHA [ ] + BETA [ ]

SIC Programming Examples
(4) Looping and Indexing: part II

SIC/XE

```
        LDS     #3          INITIALIZE REGISTER S TO 3
        LDT     #300        INITIALIZE REGISTER T TO 300
        LDX     #0          INITIALIZE INDEX REGISTER TO 0
ADDLP   LDA     ALPHA,X     LOAD WORD FROM ALPHA INTO REGISTER A
        ADD     BETA,X      ADD WORD FROM BETA
        STA     GAMMA,X     STORE THE RESULT IN A WORD IN GAMMA
        ADDR    S,X         ADD 3 TO INDEX VALUE
        COMPR   X,T         COMPARE NEW INDEX VALUE TO 300
        JLT     ADDLP       LOOP IF INDEX VALUE IS LESS THAN 300
        .
        .
        .
.                           ARRAY VARIABLES--100 WORDS EACH
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
```

GAMMA [ ] ← ALPHA [ ] + BETA [ ]

SIC Programming Examples
(5) Input and Output

Read one byte from device F1

```
INLOOP    TD      INDEV     TEST INPUT DEVICE
          JEQ     INLOOP    LOOP UNTIL DEVICE IS READY
          RD      INDEV     READ ONE BYTE INTO REGISTER A
          STCH    DATA      STORE BYTE THAT WAS READ
          .
          .
          .
```

Write one byte to device 05

```
OUTLP     TD      OUTDEV    TEST OUTPUT DEVICE
          JEQ     OUTLP     LOOP UNTIL DEVICE IS READY
          LDCH    DATA      LOAD DATA BYTE INTO REGISTER A
          WD      OUTDEV    WRITE ONE BYTE TO OUTPUT DEVICE
          .
          .
          .
INDEV     BYTE    X'F1'     INPUT DEVICE NUMBER
OUTDEV    BYTE    X'05'     OUTPUT DEVICE NUMBER
DATA      RESB    1         ONE-BYTE VARIABLE
```

SIC Programming Examples
(6) Subroutine Call

SIC

```
        JSUB    READ        CALL READ SUBROUTINE
        .
        .
        .
.                           SUBROUTINE TO READ 100-BYTE RECORD
READ    LDX     ZERO        INITIALIZE INDEX REGISTER TO 0
RLOOP   TD      INDEV       TEST INPUT DEVICE
        JEQ     RLOOP       LOOP IF DEVICE IS BUSY
        RD      INDEV       READ ONE BYTE INTO REGISTER A
        STCH    RECORD,X    STORE DATA BYTE INTO RECORD
        TIX     K100        ADD 1 TO INDEX AND COMPARE TO 100
        JLT     RLOOP       LOOP IF INDEX IS LESS THAN 100
        RSUB                EXIT FROM SUBROUTINE
        .
        .
        .
INDEV   BYTE    X'F1'       INPUT DEVICE NUMBER
RECORD  RESB    100         100-BYTE BUFFER FOR INPUT RECORD
.                           ONE-WORD CONSTANTS
ZERO    WORD    0
K100    WORD    100
```

Read 100 bytes from device F1

SIC Programming Examples
(6) Subroutine Call

SIC/XE

```
          JSUB    READ            CALL READ SUBROUTINE
          .
          .
          .
.                                 SUBROUTINE TO READ 100-BYTE RECORD
READ      LDX     #0              INITIALIZE INDEX REGISTER TO 0
          LDT     #100            INITIALIZE REGISTER T TO 100
RLOOP     TD      INDEV           TEST INPUT DEVICE
          JEQ     RLOOP           LOOP IF DEVICE IS BUSY
          RD      INDEV           READ ONE BYTE INTO REGISTER A
          STCH    RECORD,X        STORE DATA BYTE INTO RECORD
          TIXR    T               ADD 1 TO INDEX AND COMPARE TO 100
          JLT     RLOOP           LOOP IF INDEX IS LESS THAN 100
          RSUB                    EXIT FROM SUBROUTINE
          .
          .
          .
INDEV     BYTE    X'F1'           INPUT DEVICE NUMBER
RECORD    RESB    100             100-BYTE BUFFER FOR INPUT RECORD
```

Read 100 bytes from device F1

Real Machines

Two Categories

- Complex Instruction Set Computers (CISC)
  - Relative large and complicated instruction set, more instruction formats, instruction lengths, and addressing modes
  - Hardware implementation is complex
  - Examples:
    - VAX
    - Intel x86
- Reduced Instruction Set Computers (RISC)
  - Simplified design, faster and less expensive processor development, greater reliability, faster instruction execution times
  - Examples:
    - Sun SPARC
    - PowerPC

VAX Architecture

- Memory: 2^{32} bytes in virtual address space
  - consists of 8-bit bytes:
    - word: 2 bytes
    - longword: 4 bytes
    - quadword: 8 bytes
    - octaword: 16 bytes
  - can be divided into
    - System space (OS and shared space)
    - Process space (defined separately for each process)

VAX Architecture

- 32-bit registers
  - 16 general-purpose registers
    - R0-R11: no special functions
    - AP: argument pointer (address of arguments when making a procedure call)
    - FP: frame pointer (address of the stack frame when making a procedure call)
    - SP: stack pointer (top of stack in program's process space)
    - PC: program counter
  - PSL: processor status longword
  - Control registers to support OS

VAX Architecture

- Data formats
  - Characters: 8-bit ASCII codes
  - Integers:
    - 2, 4, 8, 16-byte binary numbers
    - 2's complement for negative values
  - Floating-point numbers:
    - 4 different floating-point data ranging in length from 4 to 16 bytes
  - Packed decimal data format
    - Each byte represents two decimal digits
  - Numeric format
    - To represent numeric values with one digit per byte
  - Queues and variable-length bit strings

VAX Architecture

- Instruction formats
  - Variable-length instruction format
  - Each instruction consists of
    - OP code (1 or 2 bytes)
    - Up to 6 operand specifiers (depends on instruction)

VAX Architecture

- Addressing modes
  - Register mode
  - Register deferred mode
  - Autoincrement and autodecrement modes
  - Base relative modes
  - PC relative modes
  - Index modes
  - Indirect modes
  - Immediate mode
  - etc

VAX Architecture

- Instruction set
  - Mnemonics format (e.g., ADDW2, MULL3)
    - Prefix: type of operation
    - Suffix: data type of the operands
    - Modifier: number of operands involved
  - In addition to computation, data movement and conversion, comparison, branching, VAX provides instructions that are hardware realizations of frequently occurring sequences of codes
    - Load and store multiple registers
    - Manipulate queues and variable-length bit fields
    - Powerful instructions for calling and returning from procedure

VAX Architecture

- Input and output
  - Each I/O device has a set of registers, which are assigned locations in the physical address space, called I/O space.
  - Association of these registers with addresses in I/O space is handled by memory management routines.
  - Device driver read/write values into these registers.
  - Software routines read/write values in I/O space using *standard* instructions.

Pentium Pro Architecture

- Memory: virtual memory
  - consists of 8-bit bytes:
    - word: 2 bytes
    - double-word: 4 bytes
  - can be divided into a collection of segments with different sizes
    - Code, data, and stack segments
    - Each segment can be divided into pages
    - Segment/offset address is automatically translated into a physical byte address by Memory Management Unit.

Pentium Pro Architecture

- 32-bit registers
  - 8 general-purpose registers
    - EAX, EBX, ECX, EDX: data manipulation
    - ESI, EDI, EBP, ESP: addresses
  - EIP: pointer to the next instruction
  - FLAGS: processor status and calculation results
- 16-bit registers
  - CS, SS, DS, ES, FS, GS: segment registers
- 80-bit registers
  - 8 80-bit registers for FPU

Pentium Pro Architecture

- Data formats
  - Characters: 8-bit ASCII codes
  - Integers:
    - 1, 2, 4-byte binary numbers (8-byte signed integers for FPU)
    - 2’s complement for negative values
    - Little-endian byte ordering
    - Binary coded decimal (packed or unpacked BCD)
  - Floating-point numbers:
    - Single-precision: 32 bits
    - Double-precision: 64 bits
    - Extended-precision: 80 bits

Pentium Pro Architecture

- Instruction formats
  - Variable-length instruction format (1-10 bytes)
  - Each instruction consists of
    - Optional prefixes containing flags that modify the operation of the instruction
      - E.g., repetition count, segment register specification
    - OP code (1 or 2 bytes)
    - A number of bytes for operands and addressing modes

Pentium Pro Architecture

- Addressing modes
  - Immediate mode
  - Register mode
  - Base relative mode
  - PC relative mode
  - Index mode
  - Direct mode
  - etc

Pentium Pro Architecture

- Instruction set (>400 instructions)
  - 0, 1, 2, or 3 operands
  - Register-to-register, register-to-memory, and memory-to-memory instructions
  - Special-purpose instructions that are frequently required in high-level languages, e.g., entering and leaving procedures, checking the bounds of an array

Pentium Pro Architecture

- Input and output
  - I/O instructions that transfer one byte, word, or double-word from an I/O device into register EAX, or vice versa.
  - Repetition prefixes can be used to transfer an entire string in a single operation.

UltraSPARC Architecture

- Memory: 2^{64} bytes in virtual address space
  - consists of 8-bit bytes:
    - halfword: 2 bytes
    - word: 4 bytes
    - doubleword: 8 bytes
  - can be divided into pages
    - Virtual address is automatically translated into a physical address by the UltraSPARC Memory Management Unit.

UltraSPARC Architecture

- A large register file (>100 general-purpose registers)
  - 32 bits for original SPARC, 64 bits for UltraSPARC
  - Each procedure can access only 32 registers
    - 8 global registers
    - 24 registers in overlapped *window*
- A file of 64 double-precision floating-point registers fro FPU.
- Program counter PC

UltraSPARC Architecture

- Data formats
  - Characters: 8-bit ASCII codes
  - Integers:
    - 1, 2, 4, 8-byte binary numbers
    - 2's complement for negative values
    - Big- and little-endian and byte ordering
  - Floating-point numbers:
    - Single-precision: 32 bits
    - Double-precision: 64 bits
    - Quad-precision: 80 bits

UltraSPARC Architecture

- Instruction formats
  - Fix-length instruction format (32 bits long)
    - Can speed the process of instruction fetching and decoding
  - 3 basic instruction formats
    - Call instruction
    - Branch instruction
    - Register loads and stores, and three-operands arithmetic operations
  - Each instruction consists of
    - First 2 bits: identify formats
    - OP code
    - Operands

UltraSPARC Architecture

- Addressing modes
  - Immediate mode
  - Register direct mode
  - PC relative mode only for branch instructions
  - Register indirect with displacement
  - Register indirect indexed

UltraSPARC Architecture

- Instruction set (<100 instructions)
  - Register-to-register instructions
  - Load and store instructions (only instructions that access memory)
- Instruction execution is pipelined.
- Branch instructions are delayed branches
  - Instructions immediately following the branch instruction is actually executed *before* the branch is taken.

UltraSPARC Architecture

- Input and output
  - A range of memory locations is logically replaced by device registers.
  - Device driver read/write values into these registers.
  - Software routines read/write values in this area using *standard* instructions.